Operating Systems: Internals and Design Principles

Chapter 2 Operating System Overview

Ninth Edition
By William Stallings

Operating System

- A program that controls the execution of application programs
- An interface between applications and hardware

Main objectives of an OS:

- Convenience
- Efficiency
- Ability to evolve

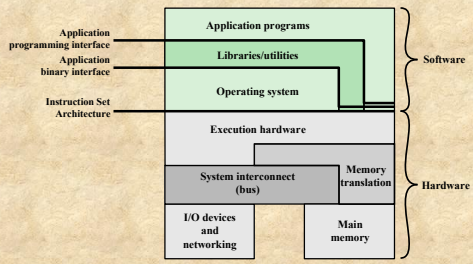


Figure 2.1 Computer Hardware and Software Structure

Operating System Services

- Program development
- Program execution
- Access I/O devices
- Controlled access to files
- System access
- Error detection and response
- Accounting

Key Interfaces

- Instruction set architecture (ISA)
- Application binary interface (ABI)
- Application programming interface (API)

The Operating System as Resource Manager

- The OS is responsible for controlling the use of a computer's resources, such as I/O, main and secondary memory, and processor execution time

Operating System as Resource Manager

- Functions in the same way as ordinary computer software
- Program, or suite of programs, executed by the processor
- Frequently relinquishes control and must depend on the processor to allow it to regain control

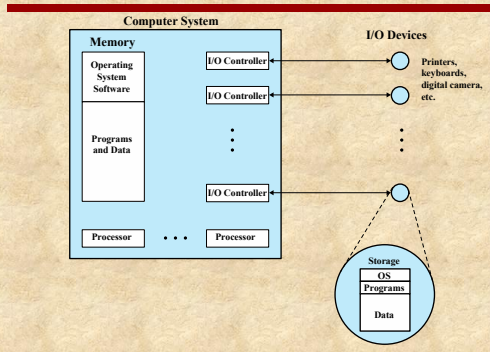


Figure 2.2 The Operating System as Resource Manager

Evolution of Operating Systems

- A major OS will evolve over time for a number of reasons:

Hardware upgrades

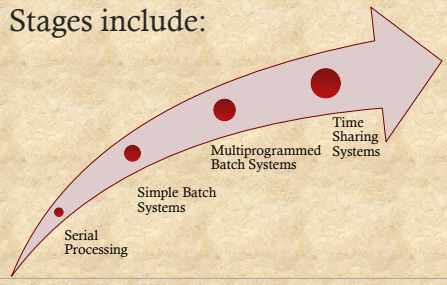
New types of hardware

New services

Fixes

Evolution of Operating Systems

- Stages include:



Serial Processing

Earliest Computers:

- No operating system
  - Programmers interacted directly with the computer hardware
- Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
- Users have access to the computer in "series"

Problems:

- Scheduling:
  - Most installations used a hardcopy sign-up sheet to reserve computer time
    - Time allocations could run short or long, resulting in wasted computer time
- Setup time
  - A considerable amount of time was spent on setting up the program to run

Simple Batch Systems

- Early computers were very expensive
  - Important to maximize processor utilization
- Monitor
  - User no longer has direct access to processor
  - Job is submitted to computer operator who batches them together and places them on an input device
  - Program branches back to the monitor when finished

Monitor Point of View

- Monitor controls the sequence of events
- *Resident Monitor* is software always in memory
- Monitor reads in job and gives control
- Job returns control to monitor

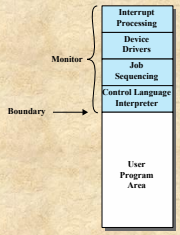


Figure 2.3 Memory Layout for a Resident Monitor

Processor Point of View

- Processor executes instruction from the memory containing the monitor
- Executes the instructions in the user program until it encounters an ending or error condition
- "*Control is passed to a job*" means processor is fetching and executing instructions in a user program
- "*Control is returned to the monitor*" means that the processor is fetching and executing instructions from the monitor program

Job Control Language (JCL)

Special type of programming language used to provide instructions to the monitor

What compiler to use

What data to use

Desirable Hardware Features

Memory protection
- While the user program is executing, it must not alter the memory area containing the monitor

Timer
- Prevents a job from monopolizing the system

Privileged instructions
- Can only be executed by the monitor

Interrupts
- Gives OS more flexibility in controlling user programs

Modes of Operation

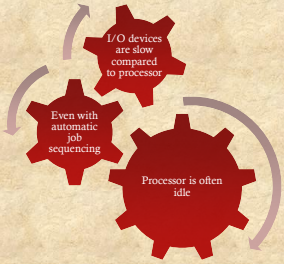
User Mode
- User program executes in user mode
- Certain areas of memory are protected from user access
- Certain instructions may not be executed

Kernel Mode
- Monitor executes in kernel mode
- Privileged instructions may be executed
- Protected areas of memory may be accessed

Simple Batch System Overhead

- Processor time alternates between execution of user programs and execution of the monitor
- Sacrifices:
  - Some main memory is now given over to the monitor
  - Some processor time is consumed by the monitor
- Despite overhead, the simple batch system improves utilization of the computer

Multiprogrammed Batch Systems

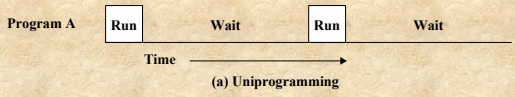


Read one record from file	15 µs
Execute 100 instructions	1 µs
Write one record to file	15 µs
TOTAL	31 µs

Percent CPU Utilization $= \frac{1}{31} = 0.032 = 3.2\%$

Figure 2.4 System Utilization Example

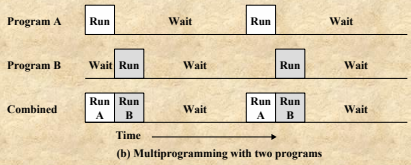
Uniprogramming



(a) Uniprogramming

The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding

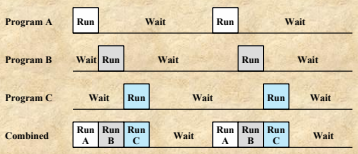
Multiprogramming



(b) Multiprogramming with two programs

- There must be enough memory to hold the OS (resident monitor) and one user program
- When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O

Multiprogramming



(c) Multiprogramming with three programs

- Also known as multitasking
- Memory is expanded to hold three, four, or more programs and switch among all of them

Multiprogramming Example

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

Table 2.1 Sample Program Execution Attributes

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Table 2.2 Effects of Multiprogramming on Resource Utilization

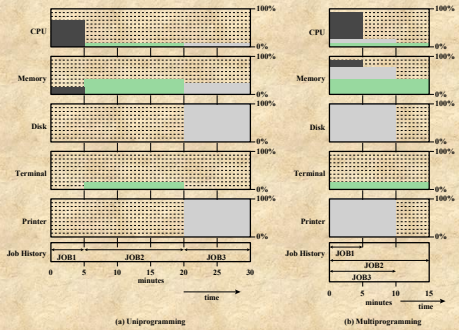


Figure 2.6 Utilization Histograms

Time-Sharing Systems

- Can be used to handle multiple interactive jobs
- Processor time is shared among multiple users
- Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Table 2.3 Batch Multiprogramming versus Time Sharing

Compatible Time-Sharing System (CTSS)

- One of the first time-sharing operating systems
- Developed at MIT by a group known as Project MAC
- The system was first developed for the IBM 709 in 1961
- Ran on a computer with 32,000 36-bit words of main memory, with the resident monitor consuming 5000 of that
- Utilized a technique known as *time slicing*
  - System clock generated interrupts at a rate of approximately one every 0.2 seconds
  - At each clock interrupt the OS regained control and could assign the processor to another user
  - Thus, at regular time intervals the current user would be preempted and another user loaded in
  - To preserve the old user program status for later resumption, the old user programs and data were written out to disk before the new user programs and data were read in
  - Old user program code and data were restored in main memory when that program was next given a turn

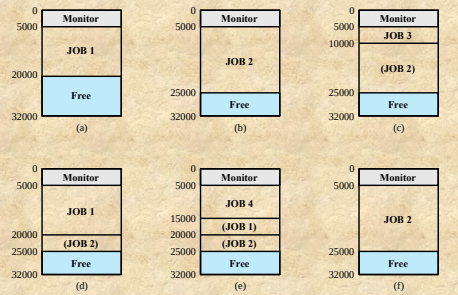


Figure 2.7 CTSS Operation

Major Achievements

- Operating Systems are among the most complex pieces of software ever developed
- Major advances in development include:
  - Processes
  - Memory management
  - Information protection and security
  - Scheduling and resource management
  - System structure

Process

- Fundamental to the structure of operating systems

A *process* can be defined as:

- A program in execution
- An instance of a running program
- The entity that can be assigned to, and executed on, a processor
- A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources

Causes of Errors

- **Improper synchronization**
  - It is often the case that a routine must be suspended awaiting an event elsewhere in the system
  - Improper design of the signaling mechanism can result in loss or duplication
- **Failed mutual exclusion**
  - More than one user or program attempts to make use of a shared resource at the same time
  - There must be some sort of mutual exclusion mechanism that permits only one routine at a time to perform an update against the file
- **Nondeterminate program operation**
  - When programs share memory, and their execution is interleaved by the processor, they may interfere with each other by overwriting common memory areas in unpredictable ways
  - The order in which programs are scheduled may affect the outcome of any particular program
- **Deadlocks**
  - It is possible for two or more programs to be hung up waiting for each other

Components of a Process

- A process contains three components:
  - An executable program
  - The associated data needed by the program (variables, work space, buffers, etc.)
  - The execution context (or "process state") of the program
- The execution context is essential:
  - It is the internal data by which the OS is able to supervise and control the process
  - Includes the contents of the various process registers
  - Includes information such as the priority of the process and whether the process is waiting for the completion of a particular I/O event

Process Management

- The entire state of the process at any instant is contained in its context
- New features can be designed and incorporated into the OS by expanding the context to include any new information needed to support the feature

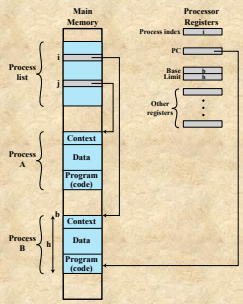


Figure 2.8 Typical Process Implementation

Memory Management

- The OS has five principal storage management responsibilities:

Process isolation	Automatic allocation and management	Support of modular programming	Protection and access control	Long-term storage

Virtual Memory

- A facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available
- Conceived to meet the requirement of having multiple user jobs reside in main memory concurrently

Paging

- Allows processes to be comprised of a number of fixed-size blocks, called pages
- Program references a word by means of a *virtual address,* consisting of a page number and an offset within the page
- Each page of a process may be located anywhere in main memory
- The paging system provides for a dynamic mapping between the virtual address used in the program and a *real address* (or physical address) in main memory

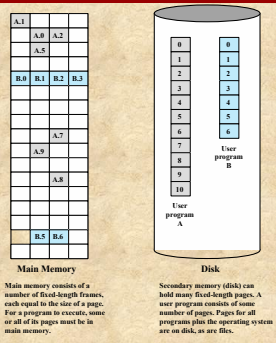


Figure 2.9 Virtual Memory Concepts

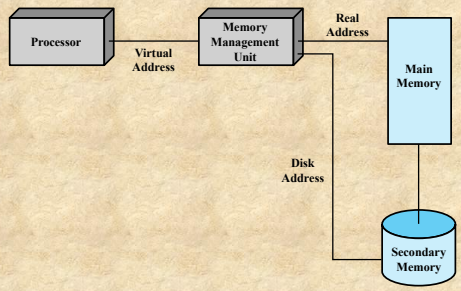
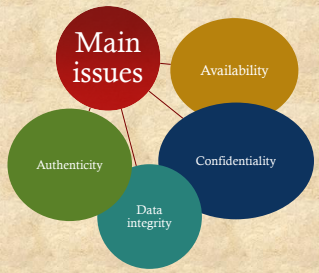


Figure 2.10 Virtual Memory Addressing

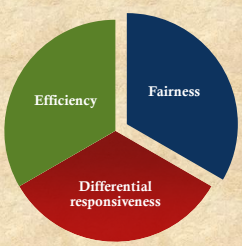
Information Protection and Security

- The nature of the threat that concerns an organization will vary greatly depending on the circumstances
- The problem involves controlling access to computer systems and the information stored in them



Scheduling and Resource Management

- Key responsibility of an OS is managing resources
- Resource allocation policies must consider:



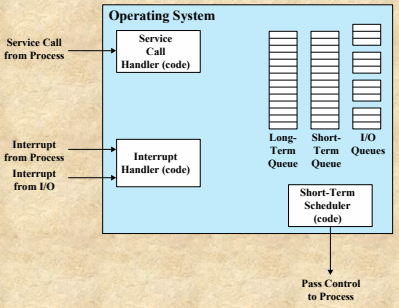


Figure 2.11 Key Elements of an Operating System for Multiprogramming

Different Architectural Approaches

- Demands on operating systems require new ways of organizing the OS

Different approaches and design elements have been tried:

- Microkernel architecture
- Multithreading
- Symmetric multiprocessing
- Distributed operating systems
- Object-oriented design

Microkernel Architecture

- Assigns only a few essential functions to the kernel:
  - Address space management
  - Interprocess communication (IPC)
  - Basic scheduling
- The approach:
  - Simplifies implementation
  - Provides flexibility
  - Well suited to a distributed environment

Multithreading

- Technique in which a process, executing an application, is divided into threads that can run concurrently

Thread	Process
Dispatchable unit of work	A collection of one or more threads and associated system resources
Includes a processor context and its own data area for a stack	By breaking a single application into multiple threads, a programmer has greater control over the modularity of the application and the timing of application-related events
Executes sequentially and is interruptible	

Symmetric Multiprocessing (SMP)

- Term that refers to a computer hardware architecture and also to the OS behavior that exploits that architecture
- The OS of an SMP schedules processes or threads across all of the processors
- The OS must provide tools and functions to exploit the parallelism in an SMP system
- Multithreading and SMP are often discussed together, but the two are independent facilities
- An attractive feature of an SMP is that the existence of multiple processors is transparent to the user

SMP Advantages

Performance	More than one process can be running simultaneously, each on a different processor
Availability	Failure of a single process does not halt the system
Incremental Growth	Performance of a system can be enhanced by adding an additional processor
Scaling	Vendors can offer a range of products based on the number of processors configured in the system

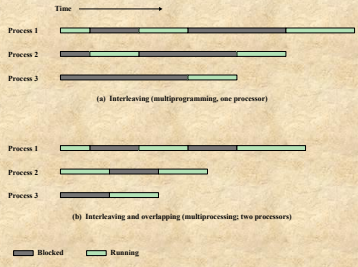


Figure 2.12 Multiprogramming and Multiprocessing

OS Design

Distributed Operating System

- Provides the illusion of a single main memory space and a single secondary memory space plus other unified access facilities, such as a distributed file system
- State of the art for distributed operating systems lags that of uniprocessor and SMP operating systems

Object-Oriented Design

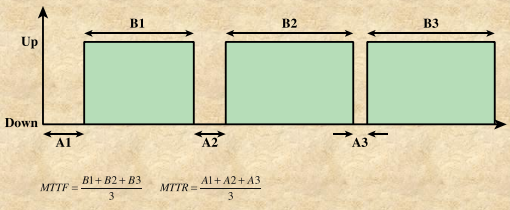
- Lends discipline to the process of adding modular extensions to a small kernel
- Enables programmers to customize an operating system without disrupting system integrity
- Also eases the development of distributed tools and full-blown distributed operating systems

Fault Tolerance

- Refers to the ability of a system or component to continue normal operation despite the presence of hardware or software faults
- Typically involves some degree of redundancy
- Intended to increase the reliability of a system
  - Typically comes with a cost in financial terms or performance
- The extent adoption of fault tolerance measures must be determined by how critical the resource is

Fundamental Concepts

- The basic measures are:
  - Reliability
    - *R(t)*
    - Defined as the probability of its correct operation up to time *t* given that the system was operating correctly at time *t=o*
  - Mean time to failure (MTTF)
    - Mean time to repair (MTTR) is the average time it takes to repair or replace a faulty element
  - Availability
    - Defined as the fraction of time the system is available to service users' requests



$$MTTF = \frac{B1 + B2 + B3}{3} \qquad MTTR = \frac{A1 + A2 + A3}{3}$$

Figure 2.13 System Operational States

Class	Availability	Annual Downtime
Continuous	1.0	0
Fault Tolerant	0.99999	5 minutes
Fault Resilient	0.9999	53 minutes
High Availability	0.999	8.3 hours
Normal Availability	0.99 - 0.995	44-87 hours

Table 2.4 Availability Classes

Faults

- Are defined by the IEEE Standards Dictionary as an erroneous hardware or software state resulting from:
  - Component failure
  - Operator error
  - Physical interference from the environment
  - Design error
  - Program error
  - Data structure error
- The standard also states that a fault manifests itself as:
  - A defect in a hardware device or component
  - An incorrect step, process, or data definition in a computer program

Fault Categories

- **Permanent**
  - A fault that, after it occurs, is always present
  - The fault persists until the faulty component is replaced or repaired
- **Temporary**
  - A fault that is not present all the time for all operating conditions
  - Can be classified as
    - Transient – a fault that occurs only once
    - Intermittent – a fault that occurs at multiple, unpredictable times

Methods of Redundancy

Spatial (physical) redundancy	Temporal redundancy	Information redundancy
Involves the use of multiple components that either perform the same function simultaneously or are configured so that one component is available as a backup in case of the failure of another component	Involves repeating a function or operation when an error is detected Is effective with temporary faults but not useful for permanent faults	Provides fault tolerance by replicating or coding data in such a way that bit errors can be both detected and corrected

Operating System Mechanisms

- A number of techniques can be incorporated into OS software to support fault tolerance:
  - Process isolation
  - Concurrency controls
  - Virtual machines
  - Checkpoints and rollbacks

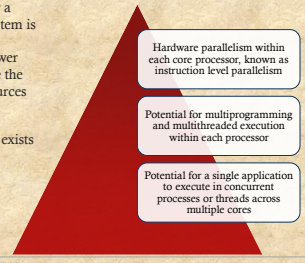
Symmetric Multiprocessor OS Considerations

- A multiprocessor OS must provide all the functionality of a multiprogramming system plus additional features to accommodate multiple processors
- **Key design issues:**

Simultaneous concurrent processes or threads	Scheduling	Synchronization	Memory management	Reliability and fault tolerance
Kernel routines need to be reentrant to allow several processors to execute the same kernel code simultaneously	Any processor may perform scheduling, which complicates the task of enforcing a scheduling policy	With multiple active processes having potential access to shared address spaces or shared I/O resources, care must be taken to provide effective synchronization	The reuse of physical pages is the biggest problem of concern	The OS should provide graceful degradation in the face of processor failure

Multicore OS Considerations

- The design challenge for a many-core multicore system is to efficiently harness the multicore processing power and intelligently manage the substantial on-chip resources efficiently
- Potential for parallelism exists at three levels:

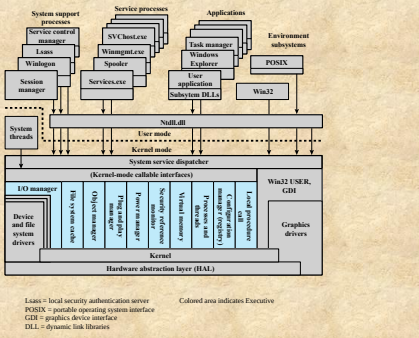


Grand Central Dispatch (GCD)

- Is a multicore support capability
  - Once a developer has identified something that can be split off into a separate task, GCD makes it as easy and noninvasive as possible to actually do so
- In essence, GCD is a thread pool mechanism, in which the OS maps tasks onto threads representing an available degree of concurrency
- Provides the extension to programming languages to allow anonymous functions, called blocks, as a way of specifying tasks
- Makes it easy to break off the entire unit of work while maintaining the existing order and dependencies between subtasks

Virtual Machine Approach

- Allows one or more cores to be dedicated to a particular process and then leave the processor alone to devote its efforts to that process
- Multicore OS could then act as a hypervisor that makes a high-level decision to allocate cores to applications but does little in the way of resource allocation beyond that



Lsass = local security authentication server
POSIX = portable operating system interface
GDI = graphics device interface
DLL = dynamic link libraries

Colored area indicates Executive

Figure 2.14 Windows Architecture

Kernel-Mode Components of Windows

- Executive
  - Contains the core OS services, such as memory management, process and thread management, security, I/O, and interprocess communication
- Kernel
  - Controls execution of the processors. The Kernel manages thread scheduling, process switching, exception and interrupt handling, and multiprocessor synchronization
- Hardware Abstraction Layer (HAL)
  - Maps between generic hardware commands and responses and those unique to a specific platform and isolates the OS from platform-specific hardware differences
- Device Drivers
  - Dynamic libraries that extend the functionality of the Executive. These include hardware device drivers that translate user I/O function calls into specific hardware device I/O requests and software components for implementing file systems, network protocols, and any other system extensions that need to run in kernel mode
- Windowing and Graphics System
  - Implements the GUI functions, such as dealing with windows, user interface controls, and drawing

Windows Executive

I/O manager
- Provides a framework through which I/O devices are accessible to applications, and is responsible for dispatching to the appropriate device drivers for further processing

Cache manager
- Improves the performance of file-based I/O by causing recently referenced file data to reside in main memory for quick access, and by deferring disk writes by holding the updates in memory for a short time before sending them to the disk in more efficient batches

Object manager
- Creates, manages, and deletes Windows Executive objects that are used to represent resources such as processes, threads, and synchronization objects and enforces uniform rules for retaining, naming, and setting the security of objects

Plug-and-play manager
- Determines which drivers are required to support a particular device and loads those drivers

Power manager
- Coordinates power management among devices

Windows Executive

Security reference monitor
- Enforces access-validation and audit-generation rules

Virtual memory manager
- Manages virtual addresses, physical memory, and the paging files on disk and controls the memory management hardware and data structures which map virtual addresses in the process's address space to physical pages in the computer's memory

Process/thread manager
- Creates, manages, and deletes process and thread objects

Configuration manager
- Responsible for implementing and managing the system registry, which is the repository for both system-wide and per-user settings of various parameters

Advanced local procedure call (ALPC) facility
- Implements an efficient cross-process procedure call mechanism for communication between local processes implementing services and subsystems

User-Mode Processes

- Windows supports four basic types of user-mode processes:

Special System Processes	• User-mode services needed to manage the system
Service Processes	• The printer spooler, event logger, and user-mode components that cooperate with device drivers, and various network services
Environment Subsystems	• Provide different OS personalities (environments)
User Applications	• Executables (EXEs) and DLLs that provide the functionality users run to make use of the system

Client/Server Model

- Windows OS services, environmental subsystems, and applications are all structured using the client/server model
- Common in distributed systems, but can be used internal to a single system
- Processes communicate via RPC

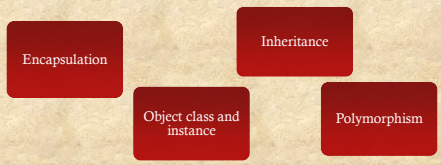
- Advantages:
  - It simplifies the Executive
  - It improves reliability
  - It provides a uniform means for applications to communicate with services via RPCs without restricting flexibility
  - It provides a suitable base for distributed computing

Threads and SMP

- Two important characteristics of Windows are its support for threads and for symmetric multiprocessing (SMP)
  - OS routines can run on any available processor, and different routines can execute simultaneously on different processors
  - Windows supports the use of multiple threads of execution within a single process. Multiple threads within the same process may execute on different processors simultaneously
  - Server processes may use multiple threads to process requests from more than one client simultaneously
  - Windows provides mechanisms for sharing data and resources between processes and flexible interprocess communication capabilities

Windows Objects

- Windows draws heavily on the concepts of object-oriented design
- Key object-oriented concepts used by Windows are:



Asynchronous Procedure Call	Used to break into the execution of a specified thread and to cause a procedure to be called in a specified processor mode.
Deferred Procedure Call	Used to postpone interrupt processing to avoid delaying hardware interrupts. Also used to implement timers and inter-processor communication
Interrupt	Used to connect an interrupt source to an interrupt service routine by means of an entry in an Interrupt Dispatch Table (IDT). Each processor has an IDT that is used to dispatch interrupts that occur on that processor.
Process	Represents the virtual address space and control information necessary for the execution of a set of thread objects. A process contains a pointer to an address map, a list of ready threads containing thread objects, a list of threads belonging to the process, the total accumulated time for all threads executing within the process, and a base priority.
Thread	Represents thread objects, including scheduling priority and quantum, and which processors the thread may run on.
Profile	Used to measure the distribution of run time within a block of code. Both user and system code can be profiled.

Table 2.5 Windows Kernel Control Objects

Traditional UNIX Systems

- Developed at Bell Labs and became operational on a PDP-7 in 1970
- The first notable milestone was porting the UNIX system from the PDP-7 to the PDP-11
  - First showed that UNIX would be an OS for all computers
- Next milestone was rewriting UNIX in the programming language C
  - Demonstrated the advantages of using a high-level language for system code
- Was described in a technical journal for the first time in 1974
- First widely available version outside Bell Labs was Version 6 in 1976
- Version 7, released in 1978, is the ancestor of most modern UNIX systems
- Most important of the non-AT&T systems was UNIX BSD (Berkeley Software Distribution), running first on PDP and then on VAX computers

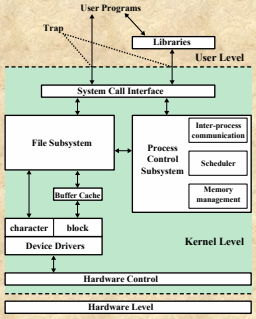


Figure 2.15 Traditional UNIX Kernel

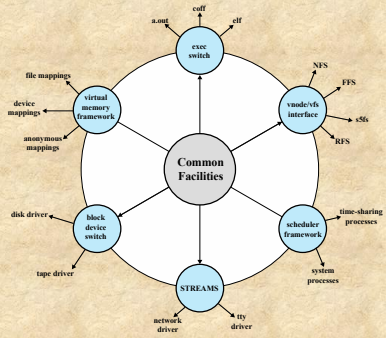


Figure 2.16 Modern UNIX Kernel

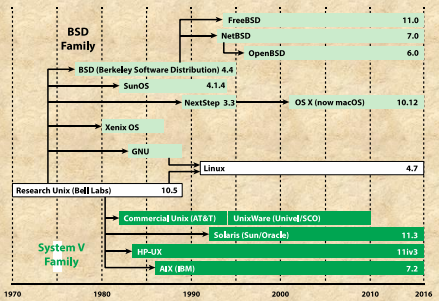


Figure 2.17 Unix Family Tree

System V Release 4 (SVR4)

- Developed jointly by AT&T and Sun Microsystems
- Combines features from SVR3, 4.3BSD, Microsoft Xenix System V, and SunOS
- New features in the release include:
  - Real-time processing support
  - Process scheduling classes
  - Dynamically allocated data structures
  - Virtual memory management
  - Virtual file system
  - Preemptive kernel

BSD

- Berkeley Software Distribution
- 4.xBSD is widely used in academic installations and has served as the basis of a number of commercial UNIX products
- 4.4BSD was the final version of BSD to be released by Berkeley
- There are several widely used, open-source versions of BSD
  - FreeBSD
    - Popular for Internet-based servers and firewalls
    - Used in a number of embedded systems
  - NetBSD
    - Available for many platforms
    - Often used in embedded systems
  - OpenBSD
    - An open-source OS that places special emphasis on security

Solaris 11

- Oracle's SVR4-based UNIX release
- Provides all of the features of SVR4 plus a number of more advanced features such as:
  - A fully preemptable, multithreaded kernel
  - Full support for SMP
  - An object-oriented interface to file systems

LINUX Overview

- Started out as a UNIX variant for the IBM PC
- Linus Torvalds, a Finnish student of computer science, wrote the initial version
- Linux was first posted on the Internet in 1991
- Today it is a full-featured UNIX system that runs on virtually all platforms
- Is free and the source code is available
- Key to the success of Linux has been the availability of free software packages under the auspices of the Free Software Foundation (FSF)
- Highly modular and easily configured

Modular Structure

- Linux development is global and done by a loosely associated group of independent developers
- Although Linux does not use a microkernel approach, it achieves many of the potential advantages of the approach by means of its particular modular architecture
- Linux is structured as a collection of modules, a number of which can be automatically loaded and unloaded on demand

Loadable Modules

- Relatively independent blocks
- A module is an object file whose code can be linked to and unlinked from the kernel at runtime
- A module is executed in kernel mode on behalf of the current process
- Have two important characteristics:
  - Dynamic linking
  - Stackable modules

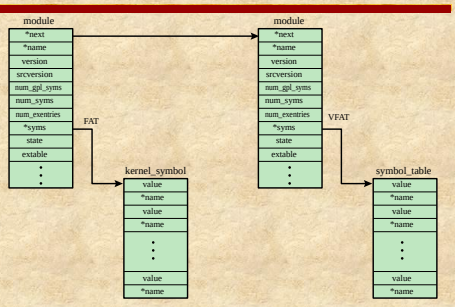


Figure 2.18 Example List of Linux Kernel Modules

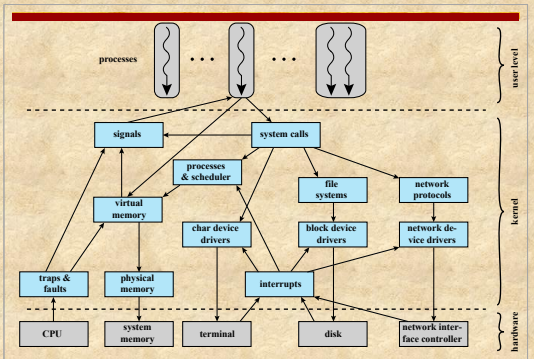


Figure 2.19 Linux Kernel Components

Linux Signals

SIGHUP	Terminal hangup	SIGCONT	Continue
SIGQUIT	Keyboard quit	SIGTSTP	Keyboard stop
SIGTRAP	Trace trap	SIGTTOU	Terminal write
SIGBUS	Bus error	SIGXCPU	CPU limit exceeded
SIGKILL	Kill signal	SIGVTALRM	Virtual alarm clock
SIGSEGV	Segmentation violation	SIGWINCH	Window size unchanged
SIGPIPT	Broken pipe	SIGPWR	Power failure
SIGTERM	Termination	SIGRTMIN	First real-time signal
SIGCHLD	Child status unchanged	SIGRTMAX	Last real-time signal

Table 2.6 Some Linux Signals

Filesystem related	
close	Close a file descriptor.
link	Make a new name for a file.
open	Open and possibly create a file or device.
read	Read from file descriptor.
write	Write to file descriptor
Process related	
execve	Execute program.
exit	Terminate the calling process.
getpid	Get process identification.
setuid	Set user identity of the current process.
ptrace	Provides a means by which a parent process my observe and control the execution of another process, and examine and change its core image and registers.
Scheduling related	
sched_getparam	Sets the scheduling parameters associated with the scheduling policy for the process identified by `pid`.
sched_get_priority_max	Returns the maximum priority value that can be used with the scheduling algorithm identified by `policy`.
sched_setscheduler	Sets both the scheduling policy (e.g., FIFO) and the associated parameters for the process `pid`.
sched_rr_get_interval	Writes into the timespec structure pointed to by the parameter `tp` the round robin time quantum for the process `pid`.
sched_yield	A process can relinquish the processor voluntarily without blocking via this system call. The process will then be moved to the end of the queue for its static priority and a new process gets to run.

Table 2.7 Some Linux System Calls (page 1 of 2)

Interprocess Communication (IPC) related	
msgrcv	A message buffer structure is allocated to receive a message. The system call then reads a message from the message queue specified by msqid into the newly created message buffer.
semctl	Performs the control operation specified by cmd on the semaphore set semid.
semop	Performs operations on selected members of the semaphore set semid.
shmat	Attaches the shared memory segment identified by shmid to the data segment of the calling process.
shmctl	Allows the user to receive information on a shared memory segment, set the owner, group, and permissions of a shared memory segment, or destroy a segment.
Socket (networking) related	
bind	Assigns the local IP address and port for a socket. Returns 0 for success and -1 for error.
connect	Establishes a connection between the given socket and the remote socket associated with sockaddr.
gethostname	Returns local host name.
send	Send the bytes contained in buffer pointed to by *msg over the given socket.
setsockopt	Sets the options on a socket
Miscellaneous	
fsync	Copies all in-core parts of a file to disk, and waits until the device reports that all parts are on stable storage.
time	Returns the time in seconds since January 1, 1970.
vhangup	Simulates a hangup on the current terminal. This call arranges for other users to have a "clean" tty at login time.

Table 2.7 Some Linux System Calls (page 2 of 2)

Android Operating System

- A Linux-based system originally designed for mobile phones
- The most popular mobile OS
- Development was done by Android Inc., which was bought by Google in 2005
- 1st commercial version (Android 1.0) was released in 2008
- Most recent version is Android 7.0 (Nougat)
- Android has an active community of developers and enthusiasts who use the Android Open Source Project (AOSP) source code to develop and distribute their own modified versions of the operating system
- The open-source nature of Android has been the key to its success

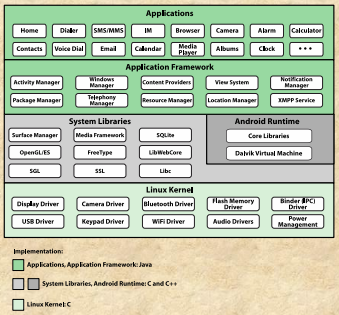


Figure 2.20 Android Software Architecture

Application Framework

- Provides high-level building blocks accessible through standardized API's that programmers use to create new apps
  - Architecture is designed to simplify the reuse of components
- Key components:

Activity Manager	Window Manager	Package Manager	Telephony Manager
Manages lifecycle of applications	Java abstraction of the underlying Surface Manager	Installs and removes applications	Allows interaction with phone, SMS, and MMS services
Responsible for starting, stopping, and resuming the various applications	Allows applications to declare their client area and use features like the status bar		

Application Framework (cont.)

Content Providers	Resource Manager	View System	Location Manager	Notification Manager	XMPP
These functions encapsulate application data that need to be shared between applications such as contacts	Manages application resources, such as localized strings and bitmaps	Provides the user interface (UI) primitives as well as UI Events	Allows developers to tap into location-based services, whether by GPS, cell tower IDs, or local Wi-Fi databases	Manages events, such as arriving messages and appointments	Provides standardized messaging functions between applications

System Libraries

- Collection of useful system functions written in C or C++ and used by various components of the Android system
- Called from the application framework and applications through a Java interface
- Exposed to developers through the Android application framework
- Some of the key system libraries include:
  - Surface Manager
  - OpenGL
  - Media Framework
  - SQL Database
  - Browser Engine
  - Bionic LibC

Android Runtime

- Most Android software is mapped into a bytecode format which is then transformed into native instructions on the device itself
- Earlier releases of Android used a scheme known as Dalvik, however Dalvik has a number of limitations in terms of scaling up to larger memories and multicore architectures
- More recent releases of Android rely on a scheme known as Android runtime (ART)
- ART is fully compatible with Dalvik's existing bytecode format so application developers do not need to change their coding to be executable under ART
- Each Android application runs in its own process, with its own instance of the Dalvik VM

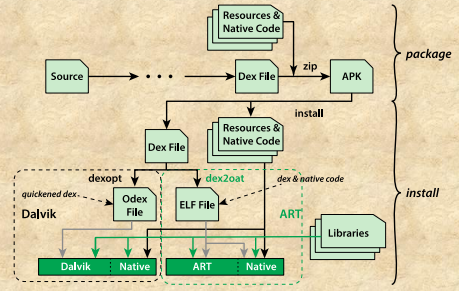


Figure 2.21 The Life Cycle of an APK

Advantages and Disadvantages of Using ART

Advantages

- Reduces startup time of applications as native code is directly executed
- Improves battery life because processor usage for JIT is avoided
- Lesser RAM footprint is required for the application to run as there is no storage required for JIT cache
- There are a number of Garbage Collection optimizations and debug enhancements that went into ART

Disadvantages

- Because the conversion from bytecode to native code is done at install time, application installation takes more time
- On the first fresh boot or first boot after factory reset, all applications installed on a device are compiled to native code using dex2opt, therefore the first boot can take significantly longer to reach Home Screen compared to Dalvik
- The native code thus generated is stored on internal storage that requires a significant amount of additional internal storage space

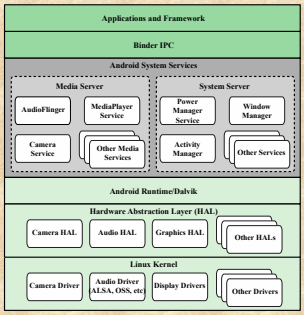


Figure 2.22 Android System Architecture

Activities

An activity is a single visual user interface component, including things such as menu selections, icons, and checkboxes

Every screen in an application is an extension of the Activity class

Activities use Views to form graphical user interfaces that display information and respond to user actions

Power Management

Alarms

- Implemented in the Linux kernel and is visible to the app developer through the AlarmManager in the RunTime core libraries
- Is implemented in the kernel so that an alarm can trigger even if the system is in sleep mode
  - This allows the system to go into sleep mode, saving power, even though there is a process that requires a wake up

Wakelocks

- Prevents an Android system from entering into sleep mode
- These locks are requested through the API whenever an application requires one of the managed peripherals to remain powered on
- An application can hold one of the following wakelocks:
  - Full_Wake_Lock
  - Partial_Wake_Lock
  - Screen_Dim_Wake_Lock
  - Screen_Bright_Wake_Lock

Summary

- Operating system objectives and functions
  - User/computer interface
  - Resource manager
- Evolution of operating systems
  - Serial processing
  - Simple/multiprogrammed/time-sharing batch systems
- Major achievements
- Developments leading to modern operating systems
- Fault tolerance
  - Fundamental concepts
  - Faults
  - OS mechanisms
- OS design considerations for multiprocessor and multicore
- Microsoft Windows overview
- Traditional Unix systems
  - History/description
- Modern Unix systems
  - System V Release 4 (SVR4)
  - BSD
  - Solaris 10
- Linux
  - History
  - Modular structure
  - Kernel components
- Android
  - Software/system architecture
  - Activities
  - Power management